LockedMe.com – Virtual Key for Repositories

This document contains:
Demonstrating the product capabilities, appearance, and user interactions.

The code for this project is hosted at https://github.com/AbhishekJha02/LockedMe.com
The project is developed by Abhishek Kumar Jha.

Step 1: Creating a new project in Eclipse

- Open Eclipse
- Go to File -> New -> Project -> Java Project -> Next.
- Type in any project name and click on "Finish."
- Select your project and go to File -> New -> Class.
- Enter **LockedMeMain** in any class name, check the checkbox "public static void main(String[] args)", and click on "Finish."

Step 2: Writing a program in Java for the entry point of the application (**LockedMeMain.java**)

```
package com.lockedme;

public class LockedMeMain {

	public static void main(String[] args) {

		// Create "main" folder if not present in current folder structure
		FileOperations.createMainFolderIfNotPresent("main");

		MenuOptions.printWelcomeScreen("LockedMe", "Abhishek Kumar Jha");

		HandleOptions.handleWelcomeScreenInput();
	}


}
```

Step 3: Writing a program in Java to display Menu options available for the user (**MenuOptions.java**)

- Select your project and go to File -> New -> Class.
- Enter **MenuOptions** in class name and click on "Finish."
- **MenuOptions** consists methods for -:

3.1. Displaying Welcome Screen

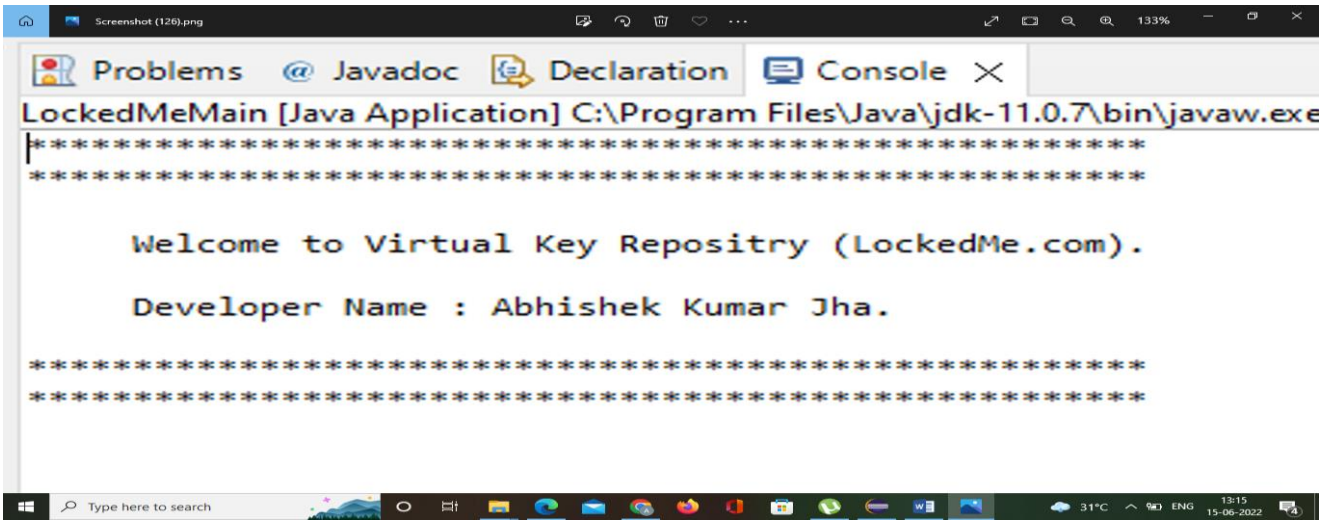
3.2. Displaying Initial Menu

3.3. Displaying Business Level Operations available

Step 3.1: Writing method to display Welcome Screen

```
public static void printWelcomeScreen(String appName, String developerName) {
              String companyDetails =
String.format("*****************************************************\n"
                  +"*****************************************************\n"
                  + "\n"
                         + "     Welcome to %s.  \n"
                  + "\n"
                  + "     Developer Name : %s.  \n"
                  + "\n"
                  + "*****************************************************\n"
                  + "*****************************************************\n",
                    appName, developerName);
                  System.out.println(companyDetails);
```

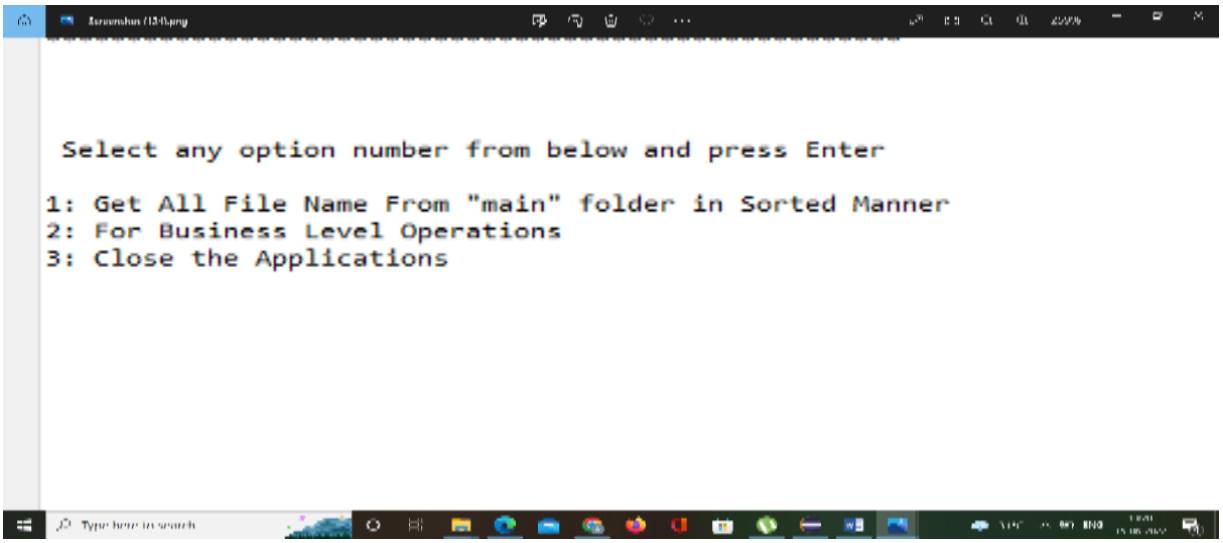
Output:



Step 3.2: Writing method to display Initial Menu

```
public static void displayMenu() {
	String menu = "\n\n Select any option number from below and press Enter \n\n"

			+ "1: Get All File Name From \"main\" folder in Sorted Manner\n"

			+ "2: For Business Level Operations\n"
			+ "3: Close the Applications\n";
		System.out.println(menu);
}
```

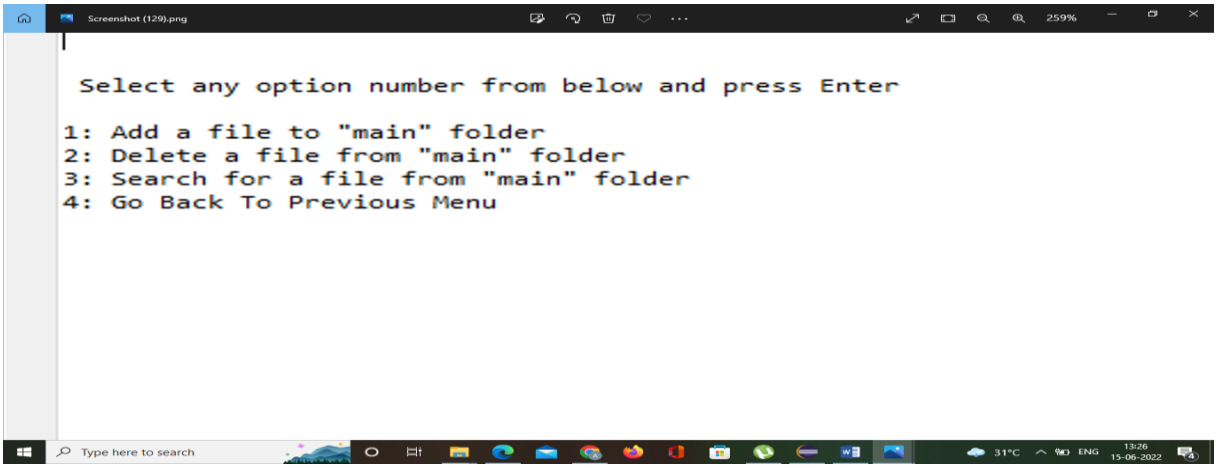
Output:



Step 3.3: Writing method to display Secondary Menu for File Operations

```
public static void BusinessLevelOperations() {
		String fileMenu = "\n\n Select any option number from below and press
Enter \n\n"
				+ "1: Add a file to \"main\" folder\n"
				+ "2: Delete a file from \"main\" folder\n"
				+ "3: Search for a file from \"main\" folder\n"
				+ "4: Go Back To Previous Menu\n" ;

		System.out.println(fileMenu);
	}
```

Output:



Step 4: Writing a program in Java to handle Menu options selected by user (**HandleOptions.java**)

- Select your project and go to File -> New -> Class.
- Enter **HandleOptions** in class name and click on "Finish."
- **HandleOptions** consists methods for -:

4.1. Handling input selected by user in initial Menu

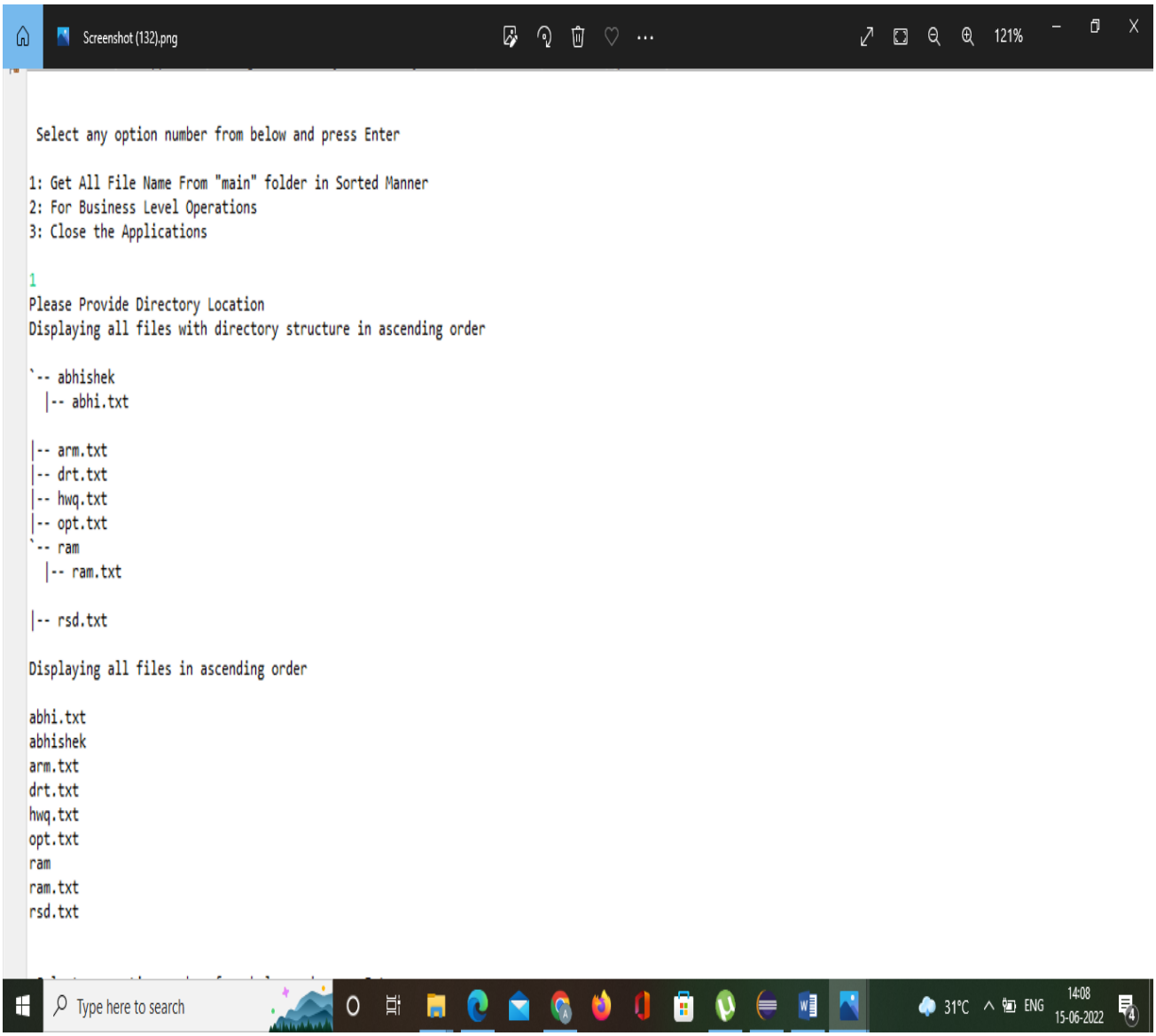
4.2. Handling input selected by user in secondary Menu for File Operations

Step 4.1: Writing method to handle user input in initial Menu

```
public static void handleWelcomeScreenInput() {
		boolean running = true;
		Scanner sc = new Scanner(System.in);
		do {
			try {
				MenuOptions.displayMenu();
				int input = sc.nextInt();

				switch (input) {
				case 1:
					System.out.println("Please Provide Directory
Location");
					FileOperations.displayAllFiles("main");
					break;
				case 2:
					HandleOptions.handleFileMenuOptions();
					break;
				case 3:
```

```
                                System.out.println("Program exited successfully.");
                                running = false;
                                sc.close();
                                System.exit(0);
                                break;
                        default:
                                System.out.println("Please select a valid option
from above.");
                        }
                } catch (Exception e) {
                        System.out.println(e.getClass().getName());
                        handleWelcomeScreenInput();
                }
        } while (running == true);
    }
```

Output:



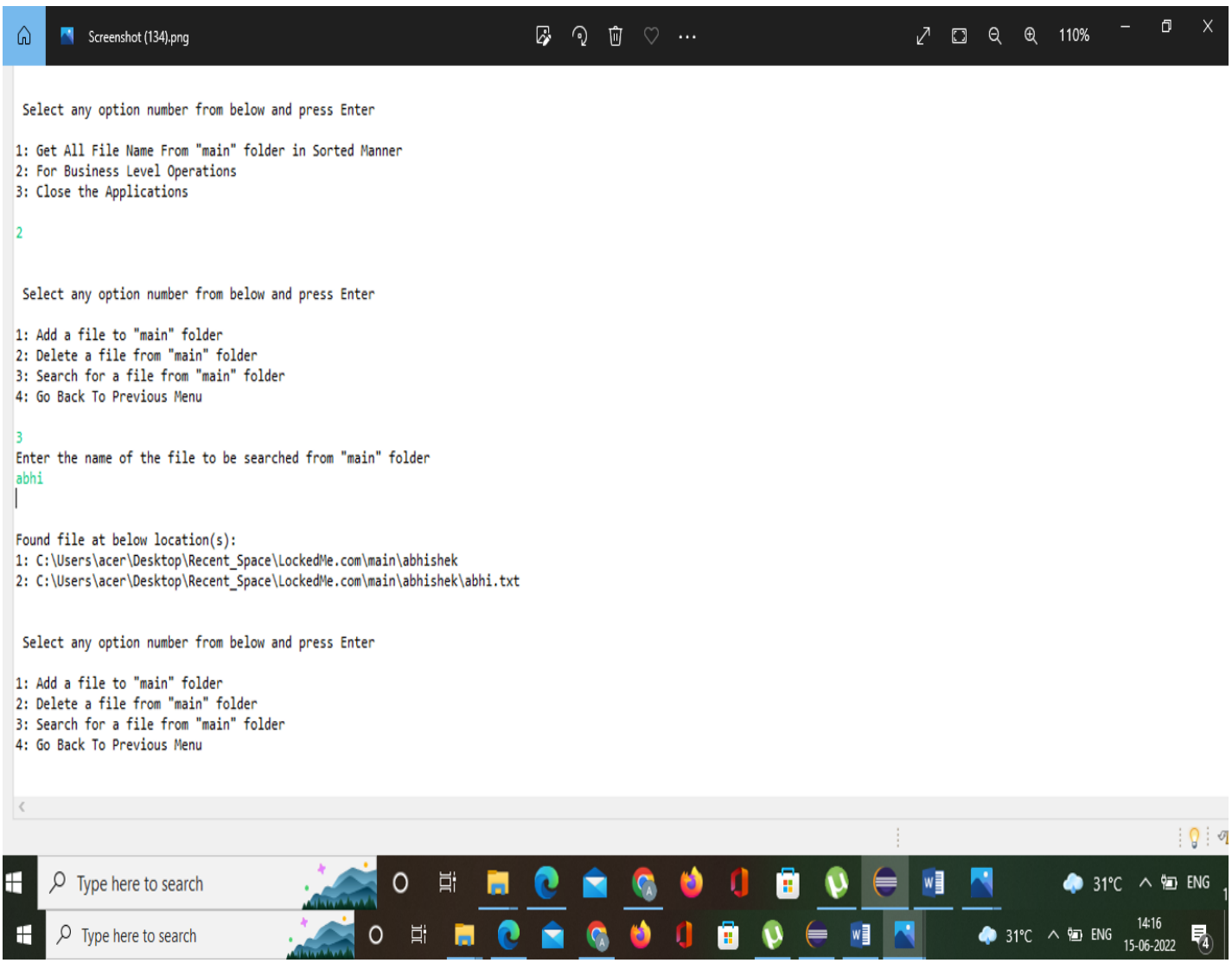
Step 4.2: Writing method to handle user input in Secondary Menu for File Operations

```
public static void handleFileMenuOptions() {
            boolean running = true;
            Scanner sc = new Scanner(System.in);
            do {
                  try {
                        MenuOptions.BusinessLevelOperations();
                        FileOperations.createMainFolderIfNotPresent("main");

                        int input = sc.nextInt();
                        switch (input) {
                        case 1:
                              // File Add
```

```
                                System.out.println("Enter the name of the file to be
added to the \"main\" folder");
                                String fileToAdd = sc.next();

                                FileOperations.createFile(fileToAdd, sc);

                                break;
                            case 2:
                                // File/Folder delete
                                System.out.println("Enter the name of the file to be
deleted from \"main\" folder");
                                String fileToDelete = sc.next();

                                FileOperations.createMainFolderIfNotPresent("main");
                                List<String> filesToDelete =
FileOperations.displayFileLocations(fileToDelete, "main");

                                String deletionPrompt = "\nSelect index of which
file to delete?"
                                            + "\n(Enter 0 if you want to delete all
elements)";
                                System.out.println(deletionPrompt);

                                int idx = sc.nextInt();

                                if (idx != 0) {

      FileOperations.deleteFileRecursively(filesToDelete.get(idx - 1));
                                } else {

                                      // If idx == 0, delete all files displayed
for the name
                                      for (String path : filesToDelete) {

      FileOperations.deleteFileRecursively(path);
                                      }
                                }


                                break;
                            case 3:
                                // File/Folder Search
                                System.out.println("Enter the name of the file to be
searched from \"main\" folder");
                                String fileName = sc.next();

                                FileOperations.createMainFolderIfNotPresent("main");
                                FileOperations.displayFileLocations(fileName,
"main");


                                break;
                            case 4:
                                // Go to Previous menu
                                return;
```

```
                        case 5:
                                // Exit
                                System.out.println("Program exited successfully.");
                                running = false;
                                sc.close();
                                System.exit(0);
                        default:
                                System.out.println("Please select a valid option
from above.");
                        }
                } catch (Exception e) {
                        System.out.println(e.getClass().getName());
                        handleFileMenuOptions();
                }
        } while (running == true);
    }
```

Output:



Step 5: Writing a program in Java to perform the File operations as specified by user (**FileOperations.java**)

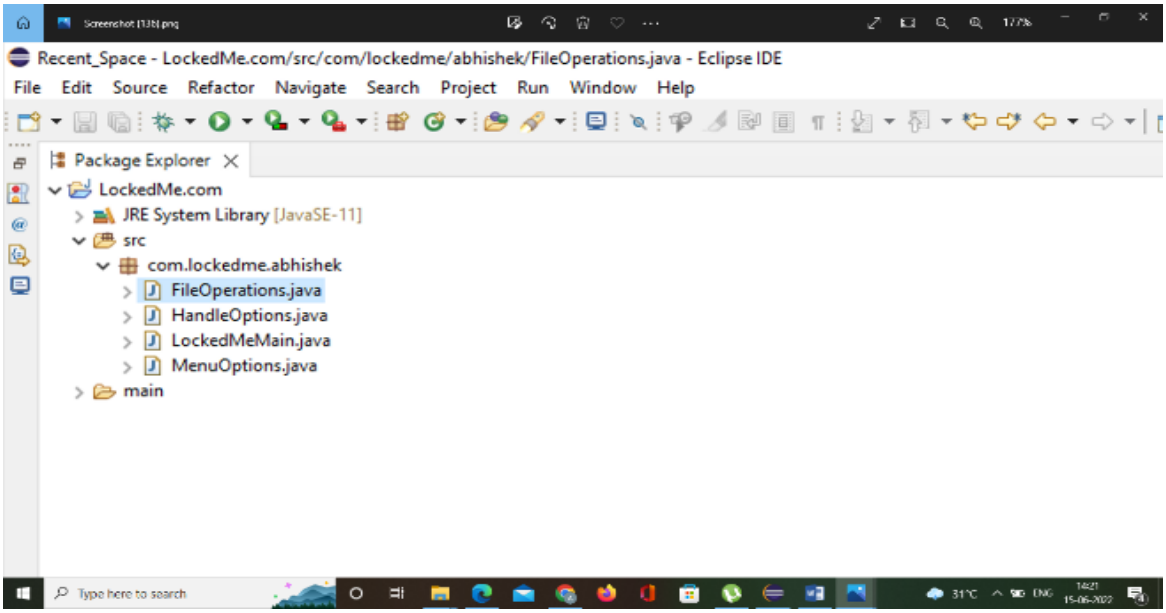
- Select your project and go to File -> New -> Class.
- Enter **FileOperations** in class name and click on "Finish."
- **FileOperations** consists methods for -:

5.1. Creating "main" folder in project if it's not already present
5.2. Displaying all files in "main" folder in ascending order and also with directory structure.
5.3. Creating a file/folder as specified by user input.
5.4. Search files as specified by user input in "main" folder and it's subfolders.
5.5. Deleting a file/folder from "main" folder

Step 5.1: Writing method to create "main" folder in project if it's not present

```
public class FileOperations {

	public static void createMainFolderIfNotPresent(String folderName) {
		File file = new File(folderName);
		// If file doesn't exist, create the main folder
		if (!file.exists()) {
			file.mkdirs();
		}
	}
```

Output:

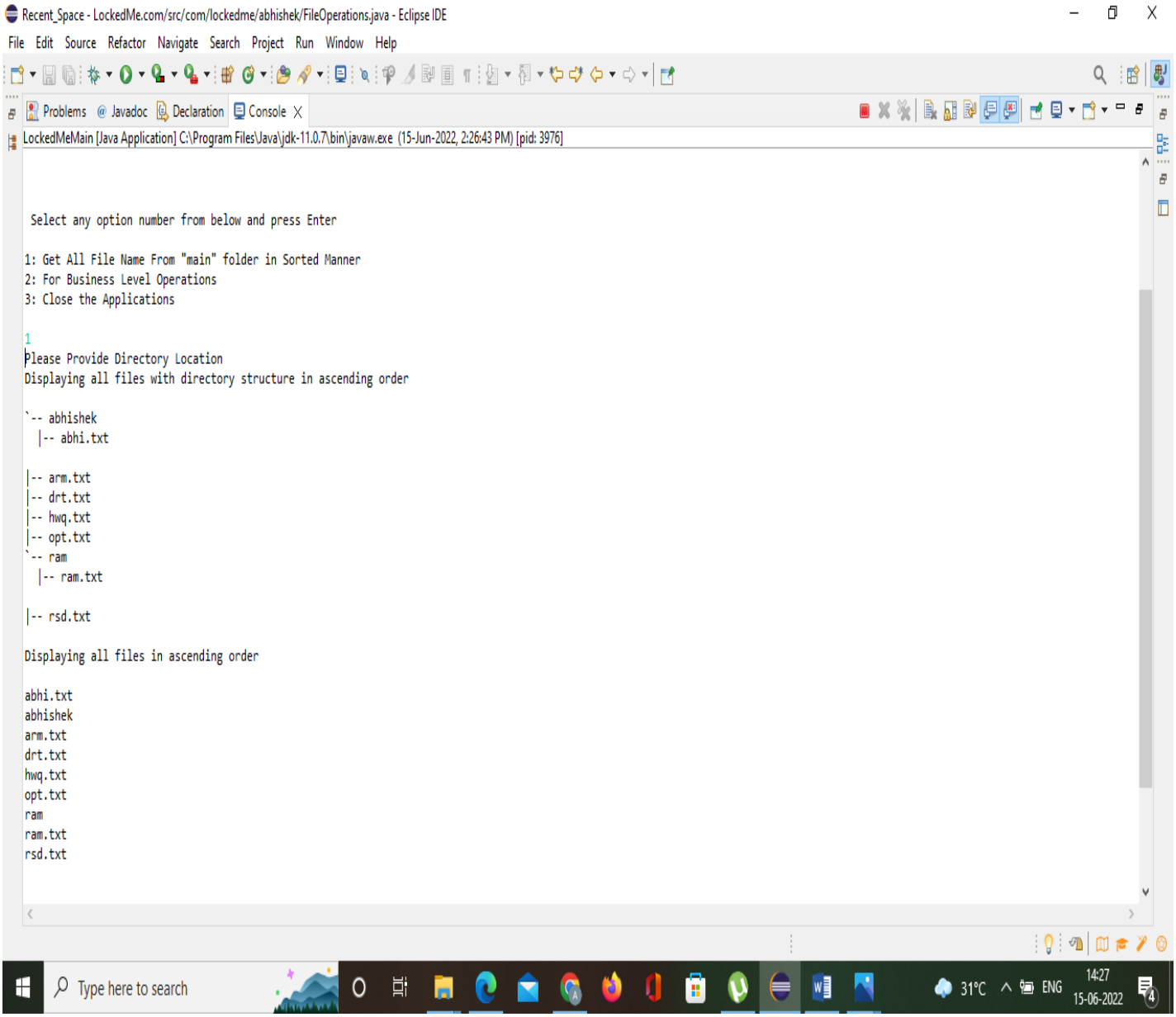


Step 5.2: Writing method to display all files in “main” folder in ascending order and also with directory structure. (“`--" represents a directory. “|--” represents a file.)

```
public static void displayAllFiles(String path) {
            FileOperations.createMainFolderIfNotPresent("main");
            // All required files and folders inside "main" folder relative to
current
            // folder
            System.out.println("Displaying all files with directory structure in
ascending order\n");

            // listFilesInDirectory displays files along with folder structure
            List<String> filesListNames = FileOperations.listFilesInDirectory(path,
0, new ArrayList<String>());

            System.out.println("Displaying all files in ascending order\n");
            Collections.sort(filesListNames);

            filesListNames.stream().forEach(System.out::println);
      }

      public static List<String> listFilesInDirectory(String path, int
indentationCount, List<String> fileListNames) {
            File dir = new File(path);
            File[] files = dir.listFiles();
            List<File> filesList = Arrays.asList(files);

            Collections.sort(filesList);

            if (files != null && files.length > 0) {
                  for (File file : filesList) {

                        System.out.print(" ".repeat(indentationCount * 2));

                        if (file.isDirectory()) {
                              System.out.println("`-- " + file.getName());

                              // Recursively indent and display the files
                              fileListNames.add(file.getName());
                              listFilesInDirectory(file.getAbsolutePath(),
indentationCount + 1, fileListNames);
                        } else {
                              System.out.println("|-- " + file.getName());
                              fileListNames.add(file.getName());
                        }
                  }
            } else {
                  System.out.print(" ".repeat(indentationCount * 2));
                  System.out.println("|-- Empty Directory");
            }
            System.out.println();
            return fileListNames;
```

```
}
```

Output:


Recent_Space - LockedMe.com/src/com/lockedme/abhishek/FileOperations.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Console
LockedMeMain [Java Application] C:\Program Files\Java\jdk-11.0.7\bin\javaw.exe (15-Jun-2022, 2:26:43 PM) [pid: 3976]
Select any option number from below and press Enter
1: Get All File Name From "main" folder in Sorted Manner
2: For Business Level Operations
3: Close the Applications
1
Please Provide Directory Location
Displaying all files with directory structure in ascending order
`-- abhishek
|-- abhi.txt
|-- arm.txt
|-- drt.txt
|-- hwq.txt
|-- opt.txt
`-- ram
|-- ram.txt
|-- rsd.txt
Displaying all files in ascending order
abhi.txt
abhishek
arm.txt
drt.txt
hwq.txt
opt.txt
ram
ram.txt
rsd.txt
Type here to search
31°C
ENG
14:27
15-06-2022

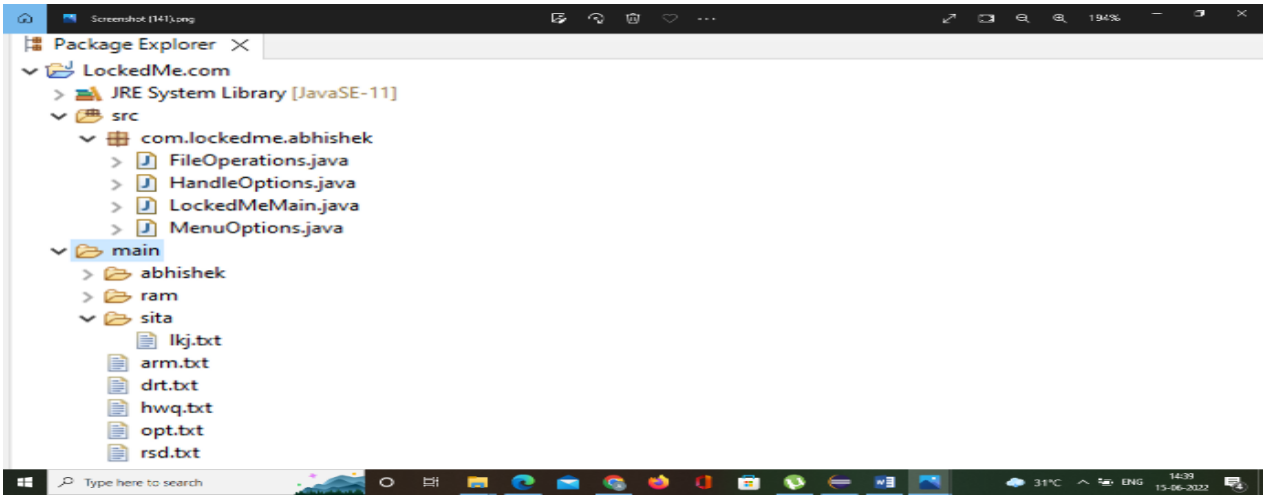
Step 5.3: Writing method to create a file/folder as specified by user input.

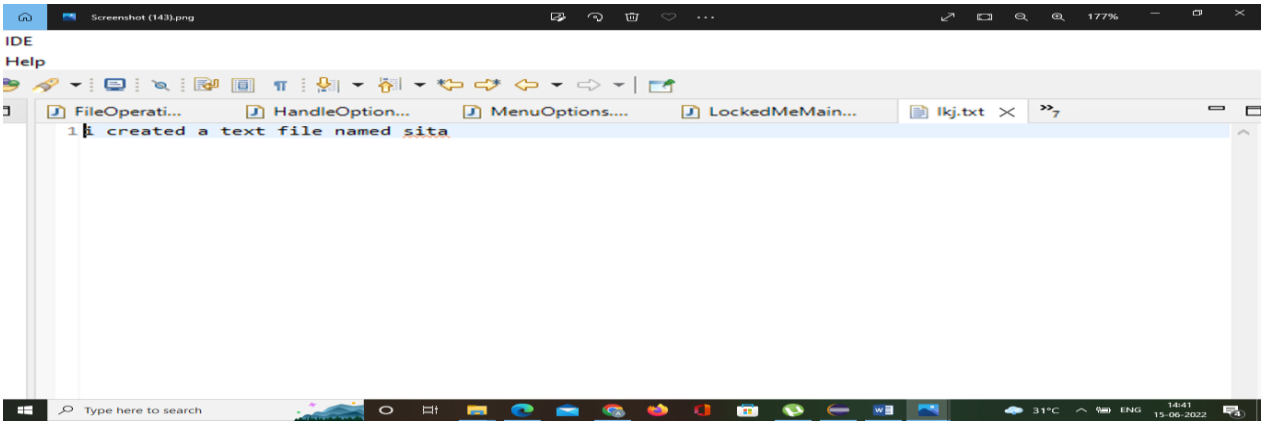
```
public static void createFile(String fileToAdd, Scanner sc) {
		FileOperations.createMainFolderIfNotPresent("main");
		Path pathToFile = Paths.get("./main/" + fileToAdd);
		try {
			Files.createDirectories(pathToFile.getParent());
			Files.createFile(pathToFile);
			System.out.println(fileToAdd + " created successfully");

			System.out.println("Would you like to add some content to the
file? (Y/N)");
			String choice = sc.next().toLowerCase();

			sc.nextLine();
			if (choice.equals("y")) {
				System.out.println("\n\nInput content and press enter\n");
				String content = sc.nextLine();
				Files.write(pathToFile, content.getBytes());
				System.out.println("\nContent written to file " +
fileToAdd);
				System.out.println("Content can be read using Notepad or
Notepad++");
			}

		} catch (IOException e) {
			System.out.println("Failed to create file " + fileToAdd);
			System.out.println(e.getClass().getName());
		}
	}
```

Output:

Folders are automatically created along with file

```
 Select any option number from below and press Enter

1: Add a file to "main" folder
2: Delete a file from "main" folder
3: Search for a file from "main" folder
4: Go Back To Previous Menu

1
Enter the name of the file to be added to the "main" folder
/sita/lkj.txt
/sita/lkj.txt created successfully
Would you like to add some content to the file? (Y/N)
y


Input content and press enter

i created a text file named sita

Content written to file /sita/lkj.txt
Content can be read using Notepad or Notepad++
```



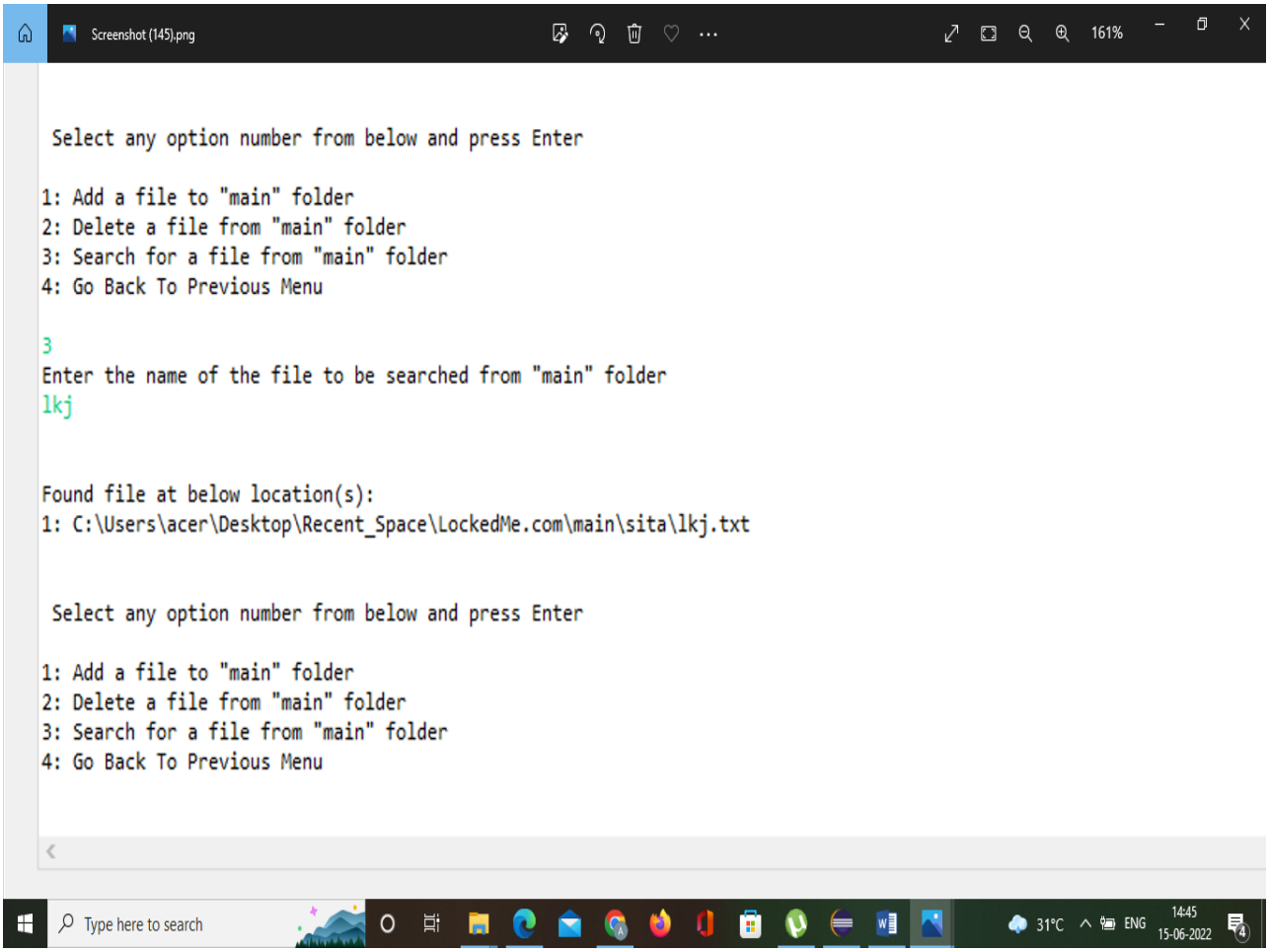


Step 5.4: Writing method to search for all files as specified by user input in “main” folder and it’s subfolders.

```
public static List<String> displayFileLocations(String fileName, String path) {
            List<String> fileListNames = new ArrayList<>();
            FileOperations.searchFileRecursively(path, fileName, fileListNames);

            if (fileListNames.isEmpty()) {
                  System.out.println("\n\n***** Couldn't find any file with given
file name \"" + fileName + "\" *****\n\n");
            } else {
                  System.out.println("\n\nFound file at below location(s):");

                  List<String> files = IntStream.range(0, fileListNames.size())
                              .mapToObj(index -> (index + 1) + ": " +
fileListNames.get(index)).collect(Collectors.toList());

                  files.forEach(System.out::println);
            }

            return fileListNames;
      }

      public static void searchFileRecursively(String path, String fileName,
List<String> fileListNames) {
            File dir = new File(path);
            File[] files = dir.listFiles();
            List<File> filesList = Arrays.asList(files);

            if (files != null && files.length > 0) {
                  for (File file : filesList) {

                        if (file.getName().startsWith(fileName)) {
                              fileListNames.add(file.getAbsolutePath());
                        }

                        // Need to search in directories separately to ensure all
files of required
                        // fileName are searched
                        if (file.isDirectory()) {
                              searchFileRecursively(file.getAbsolutePath(),
fileName, fileListNames);
                        }
                  }
            }
      }
```

Output:

All files starting with the user input are displayed along with index



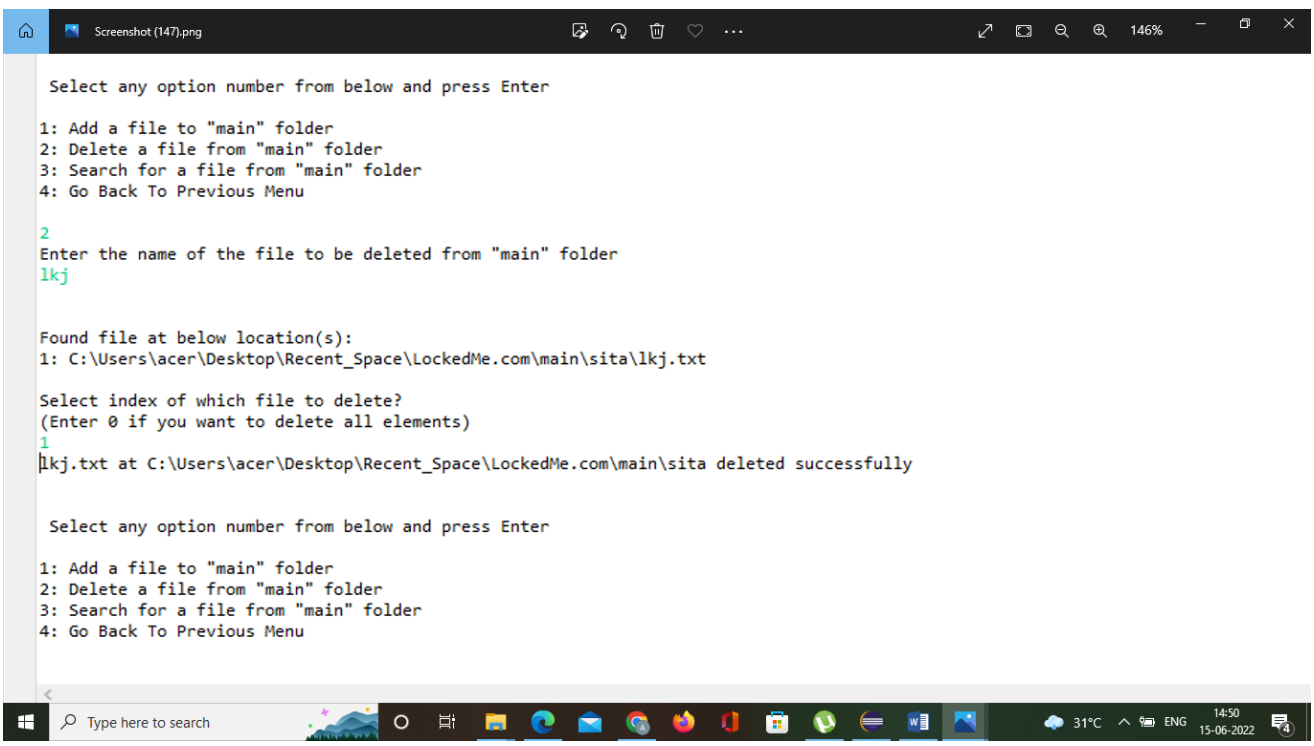
Step 5.5: Writing method to delete file/folder specified by user input in “main” folder and it’s subfolders. It uses the searchFilesRecursively method and prompts user to specify which index to delete. If folder selected, all it’s child files and folder will be deleted recursively. If user wants to delete all the files specified after the search, they can input value 0.

```
public static void deleteFileRecursively(String path) {

            File currFile = new File(path);
            File[] files = currFile.listFiles();

            if (files != null && files.length > 0) {
                  for (File file : files) {

                        String fileName = file.getName() + " at " +
file.getParent();
                        if (file.isDirectory()) {
                              deleteFileRecursively(file.getAbsolutePath());
                        }

                        if (file.delete()) {
```

```
                                System.out.println(fileName + " deleted
successfully");
                        } else {
                                System.out.println("Failed to delete " + fileName);
                        }
                }
        }

        String currFileName = currFile.getName() + " at " +
currFile.getParent();
        if (currFile.delete()) {
                System.out.println(currFileName + " deleted successfully");
        } else {
                System.out.println("Failed to delete " + currFileName);
        }
    }
}
```

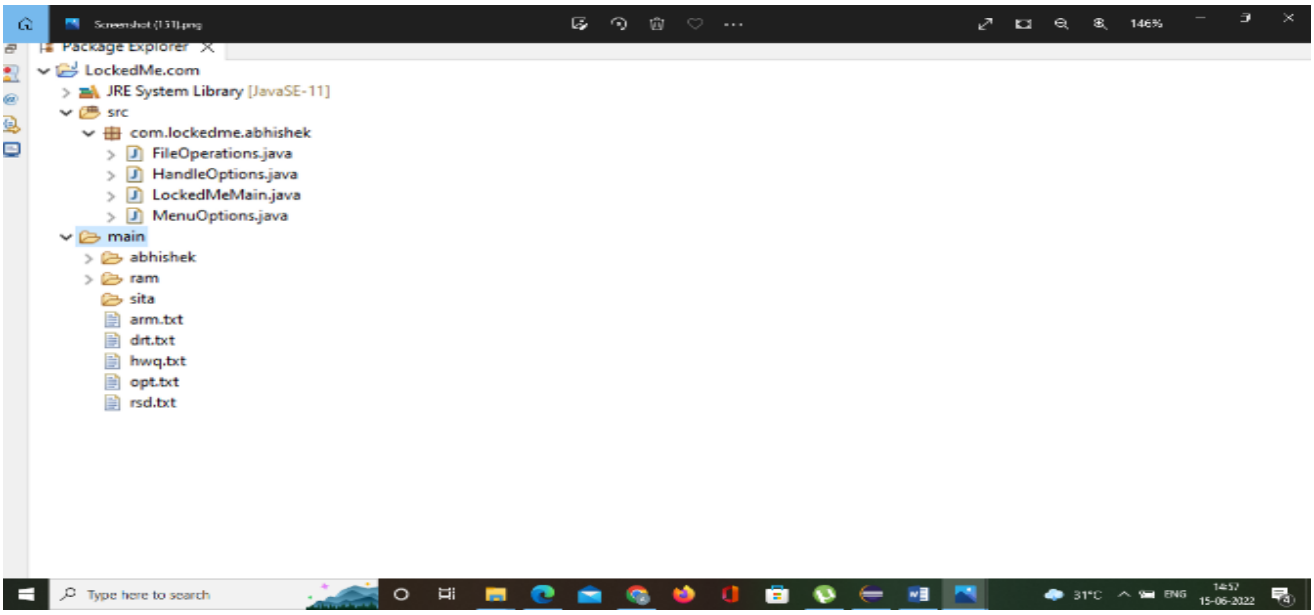
Output:

To verify if file is deleted on Eclipse, right click on Project and click “Refresh”.



Screenshot (149).png

```
 Select any option number from below and press Enter

1: Add a file to "main" folder
2: Delete a file from "main" folder
3: Search for a file from "main" folder
4: Go Back To Previous Menu

4


 Select any option number from below and press Enter

1: Get All File Name From "main" folder in Sorted Manner
2: For Business Level Operations
3: Close the Applications

1
Please Provide Directory Location
Displaying all files with directory structure in ascending order

`-- abhishek
  |-- abhi.txt

|-- arm.txt
|-- drt.txt
|-- hwq.txt
|-- opt.txt
`-- ram
  |-- ram.txt

|-- rsd.txt
`-- sita
  |-- Empty Directory
```



Step 6: Closing Application :

Output: